IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re application of:	Confirmation No.: 6544
Beloussov *et al.*	Art Unit: 2188
Appl. No.: 10/624,858	Examiner: Duc T. Doan
Filed: July 22, 2003	Atty. Docket: 2230.0340000
For: **System and Method for Using File System Snapshots for Online Data Backup**	

Declaration of Serguei Beloussov, Stanislav Protassov and Alexander Tormasov under 37 C.F.R. § 1.131

Commissioner for Patents
Washington, D.C. 20231

Sir:

The undersigned, Serguei Beloussov, Stanislav Protassov and Alexander Tormasov declare and state that,

1. We are the inventors of the above-captioned application, U.S. Appl. No. 10/624,858, filed July 22, 2003.

2. Prior to May 30, 2003, the earliest U.S. filing date of Inagaki et al., U.S. Patent Application No. 2004/0010668, we, the inventors, had completed our invention in a WTO country (specifically, one of the inventors, Serguei Beloussov, was, during the relevant time period, residing in the United States and in Singapore), as claimed in the subject application, evidenced by the following:

3. Exhibit A, submitted together with this declaration, represents a redacted version of an early draft of the present application. Exhibit A confirms that the invention was conceived prior to the filing date of Inagaki et al.

4. Exhibit B represents a collection of correspondence between the Applicants and their patent attorneys, the Los Angeles law firm O'Melveny and Myers, regarding constructive reduction to practice of the invention.

5. The documents in Exhibit B represent both correspondence and bills from the attorneys, confirming work on the preparation of the patent application, which confirms diligence in constructive reduction to practice of the invention between May 30, 2003, and the filing date of the present application, which is July 22, 2003. Specifically, the documents in Exhibit B confirm that work directed to constructive reduction to practice was performed at least on the following dates:

June 11, 2003

June 20, 2003

June 23, 2003

June 25, 2003

July 21, 2003

July 22, 2003

6. Therefore, the present invention was conceived prior to the filing date of Inagaki et al., and the inventors were diligent in working on a constuctive reduction to practice of the invention between the filing date of Inagaki et al. and July 22, 2003, the filing date of the present application.

7. As the persons signing below, we hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issue thereupon.

Date	[illegible] Serguei Beloussov
Dec. 21, 2005 Date	[illegible] Stanislav Protassov
Dec 21 2005 Date	[illegible] Alexander Tormasov

Atty Docket No. 2230.0340000

June 20, 2003

June 23, 2003

June 25, 2003

July 21, 2003

July 22, 2003

6. Therefore, the present invention was conceived prior to the filing date of Inagaki et al., and the inventors were diligent in working on a constructive reduction to practice of the invention between the filing date of Inagaki et al. and July 22, 2003, the filing date of the present application.

7. As the persons signing below, we hereby declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under § 1001 of Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the application or any patent issue thereupon.

Dec 21, 2005	[illegible]
Date	Serguei Beloussov
Dec 21, 2005	[illegible]
Date	Stanislav Protassov
Dec 21, 2005	[illegible]
Date	Alexander Tormasov

EXHIBIT A

Method of Using File System Snapshots for Online Data Backup

1. A method for backup copying of a computer file system without suspending application programs working with files by means of a file system snapshot, comprising:
   - a computer file system entirelly implemented above one or several block data storages with an ordered block set;
   - a file system cache located in the computer's operating memory and containing some data blocks for storing data in the storage and reading data by user application programs, which comprises a procedure for flushing of data blocks scheduled for write into the storage;
   - a driver program of the block data storage, which serves write and reading of data in the block mode;
   - a system of control over the procedure of data write by the driver program of the block data storage, which can determine the number of a block liable to write, send this number to the external program of backup copying, and according to the decision made by this program provide suspension and further resumption of the procedure of data blocks write into the storage;
   - a block data container, which provides temporary storage of some data intended for backup copying;
   - a system of copying blocks from the data storage into said container of block data;
   - a procedure of selecting blocks for backup copying.
2. A method of the container organization from claim 1, wherein the container may be located in the data storage external for the computer's file system, e.g., on the other disk or network server.
3. A method of the container organization from claim 1, wherein the container may be located on the same block data storage, being:
   - a separate continious area of the fixed size, allocated for using by backup copying procedures;
   - a file of the computer's file system from claim 1, acceptable through the specific application program.
4. A procedure of selecting blocks subject to backup copying from claim 1, comprising:
   - a complete list of blocks occupied by the file system including empty and dedicated block from claim 1 intended for backup copying, that is, independent of the file system type, or
   - a complete list of blocks occupied by the file system from claim 1 intended for backup copying excluding free block marked by the system as free, that is, depending on the file system type.
5. A procedure of backup copying of the file system data from claim 1, comprising:
   - setting a specific flag to a value meaning that the procedure of copying of the initial state has been started;
   -all blocks selected by the procedure from claim 4 are marked and selected for copying;
   - the procedure of backup copying, which provides backup copying for each block is started, and at the end of this process this procedure unmarks the mentioned mark for backup data;
   - after the last mark is removed from the data block, backup copying process is regarded as finished and a specific flag is set to a value meaning the procedure is over;

- the procedure of each block copying requires that at the moment of data block copying its information doesn't change;
- requests for write to the data storage, entering the driver program of the block data storage, that serves write and reading of data in the block mode from claim 1, coming from the operating system, operating system cache or application programs, are analysed by the system of control over the procedure of data writing by the driver program of the block data storage and it is checked, whether a flag meaning the procedure of backup copying is started was set and the blocks to write are marked;
- if a block to which the said write is to be performed is unmarked, then the driver program of serving block data storages continues operating and performs the required writing into the block data storage;
- if a block to which the write mentioned above is to be performed is marked, then the driver program of serving block data storages suspends writing, and performs copying of the content of the required data blocks into the container from claim 1, removing marks from the blocks required for write and permitting further write onto the disk;
- data copyied into the mentioned container to be then re-written by the procedure of backup copying instead of original set of pages;
- data in the container from claim 1 to be released after they were re-written by the procedure of backup copying;
- in case of lack of free space on the container from claim 1, all operations on writing data to this container by the mentioned driver program of serving block data storages to be suspended until the procedure of backup copy some blocks from the container from claim 1 and releases the space in the container;

6. A procedure of backup copying of the file system data from claim 5, before which the procedure of cache reset for the file system from claim 1 can be called.

Application Area

Methods of sharing files in the computer file system to provide effective multi-user access.

Level of Technology

A file system is a part of the operating system intended to provide users with a handy interface while working with data on the disk and to provide shared use of files by several users and processes.
Concept of "file system" includes:
- totality of all files on the disk,
- sets of data structures used to manage files, such as file directories, file descriptors, free and used disk space allocation tables, inode.
File systems have a rich history of development. The simpliest task of a file system is provide access to files. First file systems had such functionality. Then the tasks to raise performance with the help of caching, access markers, fault-tolerance arised.

Creation of a recoverable file system can be regarded a step forward in the evolution of file system architecture. In the past, there were two main approaches to implementing of input-output support and caching in file systems: careful write and lazy write. In file systems, developed for VAX/VMS and some other closed operating systems, an algorithm of careful write was used, and in the HPFS file system of the OS/2 operating system and in the most of UNIX file systems an algorithm of lazy write was implemented. [6]

In case of failure of an operating system or power supply interruption, input-output operations performed at that time are interrupted immidiately. Depending on what operations were performed and how far their execution advanced, such interruption may affect the integrity of the file system. In the given context violation of integrity means damage of the file system, for instance, the file name is present in the list of the directory, but the file system cannot find it and access its content. In the most serious cases damage of the file system may lead to the loss of the whole volume. A file system with careful write does not try to prevent violation of integrity. Instead it arranges records so that any system failure in the worst cases may cause unpredictable, non-critical mismatches, which the file system can eliminate at any time.
When a file system of any type receives a request for renewal of the disk content, it must perform several sub-operations before the renewal gets done. In the file system using the strategy of careful write these sub-operations always consequently write their data onto the disk. When allocating disk space, e.g., for a file, the file system first sets some bits in its bit card, and then allocates space for the file. If power supply interruption occurs right after those bits have been set, the file system with careful write loses access to that part of the disk, which was represented with the pre-set bits, but the existent data are not destroyed. Sorting write operations also means that input-output requests are performed in order of arrival. If one process alocates disk space and soon afterwords the other process creates a file, the file system with careful write completes allocating of the disk space before starting creating a file – otherwise overlap of sub-operations from two input-ouput requests might lead to violation of integrity.

The FAT file system in MS-DOS uses the through-write algorithm where renewals are performed immideately. Unlike careful write, this method does not demand input operations sorting from the file system to prevent violation of integrity.
The main advantage of file systems with careful write consists in that in case of failure the disk volume remains intact and can be further used – an intermediate launch of a volume recovery utility is not required. Such utility is needed for correction of predictable, non-destroying failures of the disk integrity, which have appeared as a result of failure, but it can be run at any time, usually at the system reboot.
However, file systems with carreful write have some imperfections – low performance, redundant non-optimized accesses to a disk, etc.

A file system with careful write sacrifices its performance for reliability. File system with lazy write increases performance thanks to the strategy of write-back caching; in the other words file changes are being written into the cache and its content is written to the disk using optimization, ususally in the background mode. Method of caching using the algorithm of lazy write provides several advantages, encreasing the performance. First, the whole number of operations of writes to the disk. Since ordered immediately performed input operations are not required, the buffer's content may vary several times before be written onto the disk. Second, the speed of servicing application requests increases sharply, because the file system may return control to the calling programm, without waiting when write to the disk is over. Finally, the strategy of lazy write ignores intermediate inconsistent states of a volume, appearing when several input-output requests overlap in time. This simplifies creation of multithreaded file system, allowing simultaneous execution of several input-output operations.
Disadvantage of the lazy write method consists in that using it you may encounter some periods when a volume acquire such an inconsistent state, that the file system is unable to correct it in case of a failure. Therefore, file systems with lazy write must always track the volume state. In general, a lazy write gives gain in performance in comparison with careful write – thanks to the higher risk and user's discomfort in case of system failure.

Recoverable file systems like Microsoft NTFS exceeds in reliability file systems with careful write at the same time reaching the performance of file systems with lazy write. Recoverable file system guarantees integrity of the volume; with this purpose a journal of changes initially created to handle transactions is used. In case of a failure of the operating system such file system recovers integrity, performing the recovery procedure on basis of information from the journal file. Such file system registers all operations of write to the disk in the journal ant recovery takes several seconds independently of the volume size. The recovery procedure in the recoverable file system is presize and guarantees the return of the volume into the consistent state. Inadequate results of recovery, typical for file systems with lazy write are impossible here.
High reliability of the recoverable file system has its disadvantages. At each transaction modifying the volume structure, it is required to enter one record per each transaction sub-operation into the journal file. The file system reduces expenses of protocolling thanks to integration of journal file records into packets: for one input-output operation several records are added to the journal at once. Besides that the recoverable file system may use optimization algorithms used by file systems with lazy write. It can even increase intervals between records of cache to the disk, because the file system can be recovered, if a falure occurred before modifications were copied from cache to the disk. Such growth in performance compensates and often even overbalances expenses for protocolling of transactions.

Neither careful, nor lazy write can guarantee protection of user data. If a system failure occurred at the moment when an application performed recording to a file, then the file may be lost or destroyed. Moreover, the failure may damage the file system with lazy record having destroyed existing files or even made all information on the volume unavailable. The recoverable file system uses the strategy increasing its reliability in comparison with traditional file systems. As an example let's consider the NTFS file system. First, recoverability of NTFS guarantees that the structure of the volume won't be destroyed because in case of the system failure all files will remain available. Second, though NTFS does not guarantee safety of user data in case of the system failure, because some modifications in cache can be lost, applications can use advantages of write-through and NTFS cache reset to guarantee that modifications of files will be written to the disk at the required time. Both write-through (forced immediate write to the disk) and cache reset (forced write of the cache content to the disk) are quite effective operations. NTFS do not require additional input-output to write modifications of several various data structures of the file system to the disk, because changes of these structures are registered in the journal file (during one write operation); if a failure occurred and the cache content was lost, modifications of the file system can be recovered using information from the file journal. Moreover, NTFS unlike FAT guarantees that right after the write-through or cache reset operation user data will stay safe and will be available even if the system failure occures afterwords.
NTFS supports recovery of the file system using the concept of atomic transaction. The idea of atomic transactions consists in that some operations on data storages called transactions are performed using a principle: everything or nothing. Single changes on the disk composing a transaction are performed atomically – that is, during the transaction all required changes are to be moved to disk. If the transaction is interrupted by a file system failure, modifications done by the current moment are cancelled. After backoff the database returns to the initial consistent state where it was before the transaction began.
It is necessary to mention that journaling is not a panacea against failures.
You open a file and a big volume of data into it. In the middle or write a failure occures and reboot, and after recovery the file is empty. All information that the user wrote into the file since it was open, disappeared. Thus, journaling file systems are dedicated not for recovery of data at any price, but for support of non-contradiction of metadata of the file system at the moment of failure. Therefore, such system works as follows: you open a file and if it opens succesfully, the file system notes opening in its journal by recording a transaction. Then you start writing. But the file system does not record copies of these data. And when after a failure recovery takes place, the backpff to the latest succesfull transaction occurs – before opening of a new empty file.
Among journaling file systems it is necessary to mention ReiserFS, JFS, XFS, ext3, NTFS. Sometimes for optimization of the journaling file system the journal is relocated to another independent device to provide asinochronous access. The performance of the system in this case increases by 30-50%. Not all systems have a journal taken out. Among mentioned only XFS and ReiserFS with installed patches.
Below are some specific features of some file systems.
XFS was created at the beginning of 90th (1992-1993) by Silicon Graphics (now SGI) for multimedia computers with OS Irix. The file system was oriented to very big files and file systems. The peculiarity of this file system is the journal construction – some metadata of the file system itself are written to the journal so that the whole recovery process is reduced to copying of these data from the journal to the file system. The size of the journal is set when the system is created and cannot be less than 32 Mb.

JFS is a journaling file system was created by IBM for OS AIX. Further is was ported to OS/2. A Linux version also exists. The journal size is about 40% of the file system size, but not bigger than 32 Mb. The peculiarity of this file system consists in "aggregates" – this file system may contain several segments including the journal and data, and each of such segments can be mounted separately.
Project ReiserFS has started at 90th, the first prototype was called TreeFS and the system exists only for OS Linux.
Ext3 is journaling superstructure over ext2 – the main and the most reliable file system OS Linux. At present, mainly RedHat develops this system. Its advantage is that it does not change the internal structure of ext2. The file system ext3 can be created out of ext2 by running the program of journal creation. Then this file system can be mounted using the driver of ext2. And then again using the ext3 driver, and create the journal.

The evolution of file systems demonstrates that fault-tolerance and recoverability of file systems after failures are very important. To provide the maximum reliability it is necessary to periodically copy all file system as a immediate cast of the file system – snapshot. By its functionality snapshot is very close to the journal of a recoverable file system, as they both provide backoff of the system to the integral state. At that snapshot guarantees full data recovery, but requires high expenses for creation and storage.
Simple mechanism of snapshot creation consists in sector by sector copying of the whole file system – that is, service information and data. Though if the file system is currently active, then during copying files can be modyfied – some files can be open for writing or locked, etc. In the simplest case the file system can be suspended for some time and during that time a snapshot is recorded. Of course, such approach cannot be applied to servers where uninterruptible activity of the file system is to be provided.
Modern file systems provides mechanisms to create snapshots without interrupting the operation of the file system. Let's consider an example based on the file system Episode [7]. The given file system operates on 'filesets' – logical elements representing a linked tree. In Episode, a 'fileset' is an object of administration, replication and reservation. The design of Episode allows disposal of several 'filesets' on one partition. To create snapshots the mechanisms of fileset cloning are used. The filset clone is a snapshot, at the same time it is a usual fileset, which shares data with the original fileset thanks to the copy-on-write technics. The cloned fileset is available for reading only, it is always placed on the same partition as the original fileset (available for reading and write). Clones are created very quickly, and the most important, without interrupting access to data being copied. Cloning is done thanks to cloning of all 'anodes' to 'filesets'. Here, 'anode' is an analogue of 'inode' in BSD with some minor distinctions. After copying each 'anode' they both (new and old) pont to the same data block. But the reference to the disk in the original 'anode' acquire the flag COW (copy-on-write), so that during block modification a new data block is created, at that the flag COW is removed from it.
As a result we get a mechanism allowing sorting in time changes happening in the file system. It can be achieved thanks to that all modifications done in the file system or in any part of it during a given period of time are written to a separate tree. Such separate trees being sorted in time represent a kind of full version of the file system modifications. Thus, to find the file state at a given moment it is enough to search sequently the required file in the tree closest in time, if it was not found there – in the previous tree, etc.
To illustrate this idea in more detail, consider one more example – implementation of snapshots in the file system WAFL WAFL (Write Anywhere File Layout) [5], which was designed for network file servers. Refer to the corresponding patent reference below in this article (patent #2). The main purpose of WAFL algorithms and data structures is to support

snapshots, which are here defined as a "read-only" file system clone. To minimize the disk space taken by the snapshot, WAFL uses the copy-on-write technology. Also, snapshots in WAFL are used to exclude the necessity to check the file system integrity after a failure, which allows the file server to start quickly.
WAFL creates and deletes snapshots automatically according to a schedule, at that it keeps up to 20 copies of snapshots to provide access to old files. The copy-on-write techology is used to prevent dubling of disk blocks (in the snapshot and active file system). Only on condition that the block in the file system is modified, the snapshot containing this block will start taking the disk space.
Users get access to snapshots via NFS. An administrator can use snapshots to create backup copies independently of the file system operation.
WAFL stores metadata in files. The three files of metadata are used: file 'inode', containing 'inode' for the file system, block map file, which identifies spare blocks and inode file map, identifying a spare inode. We use the term "map", not "bitmap", as these files use more than one bit for each record.

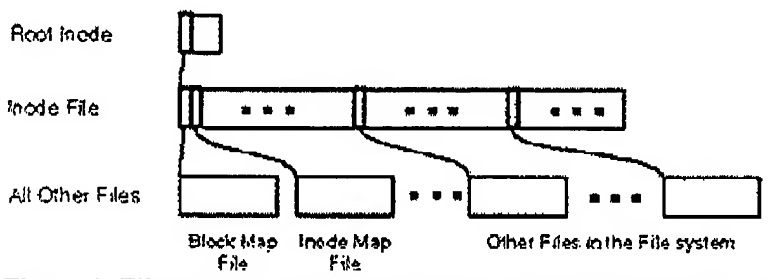


Figure 1. File system WAFL represents a tree of blocks with the root inode, pointing to the file inode (on top). Metadata and usual files are placed below.

Storage of metadata in files allows WAFL to write blocks of metadata to any palce on the disk. Such design allows using the copy-on-write technology on write during creation of snapshots. For that WAFL is to be able to write all data including metadata to a new place on the disk, excluding re-writing of old data. If WAFL could store data to any fixed place on the disk, such process wouldn't be possible.
Consider the structure of the file system WAFL.
It is better to represent the structure of WAFL as a tree of blocks. As shown on Figure 1, the root inode is in the root of the tree. The root inode is a specific inode describing the file inode. The file inode contains an inodes, describing other files in the file system, inluding files of the block map and inode map. Data blocks of all files form leaves of the tree.
Figure 2 is a more detailed version of Figure 1. It shows how files are composed of separate blocks and big files have additional links between inodes and real data blocks. For loading WAFL is to find the root of the tree, and so a single exclusion from the rule "write to any place" is the block containing the root inode. It should be located in the fixed place on the disk.

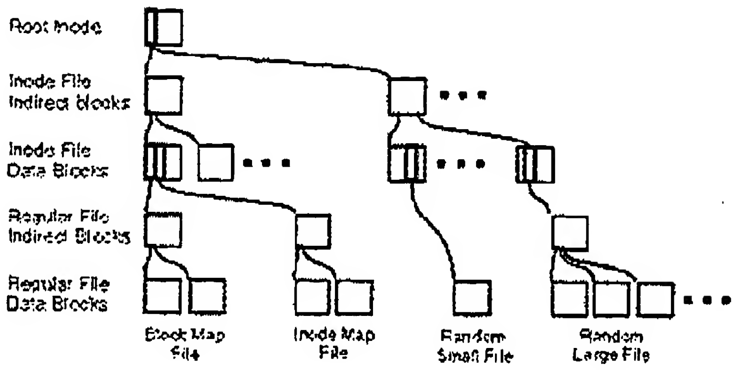


Figure 2. More detailed presentation of WAFL as a tree of blocks.

Let's discuss creation of snapshots in WAFL. To create a virtual copy of tree of blocks, WAFL simply copies the root inode. This process is depicted on Figure 3. Figure 3a is a simplified schema of the file system from Figure 3, here internal nodes of the tree like inodes and indirect blocks are omitted. Figure 3b shows how WAFL creates a new snapshot via copying the root inode. The copied inode becomes root in the tree of blocks and it represents the snapshot the same way as the root inode represents the real file system. When the snapshot's inode is being created, it points to the same disk blocks as the root inode, and so a new snapshot does not take additional disk space (excluding the space taken to create the snapshot's inode).
Figure 3c depicts the situation when a user modifies the data block D. WAFL writes new data to the block D' on the disk and modifies the pointer to the new block in the active file system. The snapshot as before points to the old block D which remained unmodified on the disk. As files are modified or deleted in the active file system, the snapshot refers to the growing amount of blocks, no longer belonging to the active file system. As well as the snapshot will take more and more disk space.

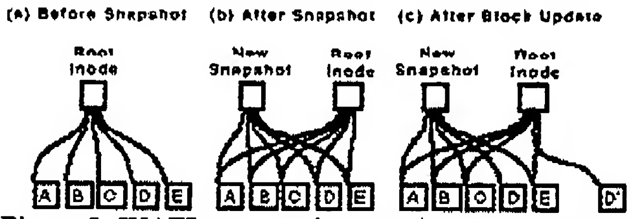


Figure 3. WAFL creates the snapshot dubling the root inode, describing the inode file. WAFL does not change snapshot's blocks, because it copies new data to the new place on the disk.

Let's compare WAFL snapshots with Episode snapshots. In Episode, instead of copying the root inode, a copy of the whole inode file is created. This significantly loads the disk subsystem and consumes a lot of disk space. For example, 10Gb file system with one inode per each 4 Kb of disk space will allocate 320 Mb for inode. In such file system creation of a snapshot through copying of the inode file will generate 320 Mb of disk traffic and take 320 Mb of disk space. Creation of ten such snapshots will take nearly one third of free disk space, taking into account that modification of blocks was performed. Copying the root inode, WAFL quickly creates a snapshot without charging the disk subsystem. It is

important, because WAFL creates snapshots every several seconds to implement a mechanism of recovery after failures.
Figure 4 shows transition from Figure 3b to 3c in more details. When a disk block is modified all its content is relocated to a new place, accordingly, the parent's block also has to be modified to be pointed to a new place. As well as the parent's parent in its turn also has to be re-written to a new place and so on.

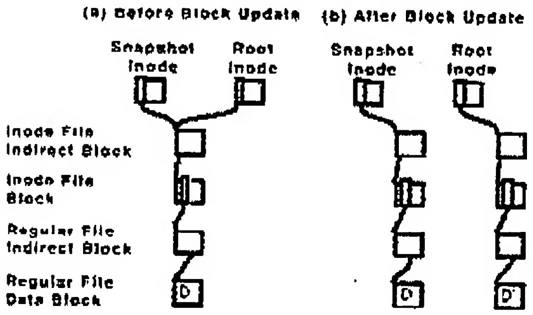


Figure 4. To write a block to a new place, the pointer of the block's parent is also should be updated, thus parents will be re-written to a new place.

WAFL would be a very slow file system, if it would write several blocks at each modification. Instead of that WAFL is cashing several hundreds modifications before write. During write WAFL allocates disk space for all data in cache and performs a disk operation. As a result blocks often modified, like indirect blocks or inode file blocks are written once during cache reset and not each time when the data are modified.

[redacted] REDACTED

[redacted]

PAGE
REDACTED

[redacted]

Problem

It is a problem in computer systems and data storage subsystems to perform the data file copy operation in a manner that minimizes the use of processing resources and data storage space in memory. In the worst case this operation consumed twice the amount of memory and also required the intervention of the processor.

[redacted]

[redacted]

[redacted]

Another problem with the incremental copy process is that this process can bypass security policies that block unauthorized data access. For example, a certain group of users may have access to a data file now used for the copy data file. The copy data file is being written though with its file access permissions have not yet been updated. Thus, the access permission process is not synchronized with the copy process and users can access files for which they are not authorized.

[redacted]

Solution

The above described problems are solved and a technical advance achieved by the present data file storage management system for snapshot copy operations. The snapshot copy updates to the contents of the first level of the two level mapping table are stored on the backend data storage devices to provide a record of the snapshot copy operation which can be used to recover the correct contents of the mapping table. This record of the snapshot copy operation remains valid even though the physical location of a copied data file instance is subsequently changed. Furthermore, the physical storage space holding the updated portions of the first level of the two level mapping table can be managed using techniques like those used to manage the physical storage space holding data file instances. In addition, the synchronization of the snapshot copy operation with the reading and writing of data to the original and copy data files is maintained by detecting accesses to the original data file or the copy data file during the time that the snapshot copy process is being executed and the mapping table is being updated. Mapping table updates resulting from the snapshot copy operation are delayed until all mapping table updates resulting from earlier data file write operations have been completed and any attempt to update the mapping table to reflect data written to the original data file or the copy data file that occurs after initiation of the snapshot copy operation must wait until the first set of mapping table pointers have been updated.

The present data file storage management system for snapshot copy operations is implemented in a dynamically mapped virtual data storage subsystem to maintain data file copy integrity in snapshot copy operations. The data storage subsystem is connected to at least one processor and functions to store data files for the processor in the backend data storage devices that are part of the data storage subsystem. The processor assigns a virtual track address to a data file which is transmitted to the data storage subsystem for storage in an allocated physical storage location in the backend data storage devices. The assignment of a physical storage location on the backend data storage devices is effected by a controller contained within the data storage subsystem, which defines the correspondence between the processor assigned virtual track address and the logical address of the stored data file. A mapping table translates the virtual track address into a logical address which identifies the location of the data file on a physical disk drive. The location of the data file changes as the data storage subsystem free space collection process moves the data file to create free space into which new data can be written. It is therefore insufficient to store the translation from a virtual address to a logical address as a means of preserving a record of the mapping table updates, since the free space collection process changes the physical location of the data file but does not update these translations. The data files stored on the disk drives must therefore contain information that is independent of the logical address at which the data file presently being copied is stored. This enables the disk stored information to remain valid even though the physical location of the data file may change over time. This is accomplished by the use of a two level mapping architecture. The first level of mapping tables maps a virtual address to an immutable name which identifies a unit of data, such as a virtual track. The second level of mapping tables maps the immutable name to a given logical address. The snapshot copy operation operates on the first level of the mapping table to create multiple copies of the virtual track, thereby eliminating the need to associate with each virtual track address a mapping table entry which contains a logical address.

In a snapshot copy operation, to provide the illusion of there being two independent copies of the data, any write to one of the copies must leave the other copy unchanged. In a log structured (journaling) file system, the writing of data or changes to a data file results in the data file being written to a different location in memory. If the original data file is accessible by a plurality of virtual addresses, it must remain in its original form, while the changes to this data file must be reflected in a new copy of the data file. The mapping table must therefore be updated in such a manner that the virtual address of the new data file points to this new copy of the data file, while the remaining virtual addresses point to the original data file location.

The problem of providing point in time copy semantics for the copy operation is solved by completing data file accesses that are already in progress before initiating the mapping table updates performed by the snapshot copy process and detecting accesses to the original data file or the copy file during the time that the copy process is being executed and the mapping table is being updated. The table updates performed at the copy process are delayed until the completion of the data file accesses that preceded the command that initiated the copy process and any attempt to update the mapping table to reflect data written to the original data file or the copy data file that occurs after initiation of the snapshot copy process must wait until the first set of mapping table pointers have been updated. This ensures that the data file access operations will be completed before the initiation of the copy operation and all further operations on the file will be performed after the copy operation is over. The use of a fault tolerant cache allows this write delay to be hidden from the processor. Any request to read data from the copy data file received before the mapping table pointers have been updated is redirected to the original data file to ensure that the data file read operation. Thus, the present file system ensures the reliability of the mapping table data and also performs an interruptible operation of the file system.

[redacted]t.
It is an object of the present invention to provide a computer system which can be directly applied to an existing computer system or file system, can efficiently take a snapshot, and can easily restart a system using a disk image of an arbitrary snapshot.
For that a computer system, which comprises a file system for managing accesses including an update access from an application program to files stored in a nonvolatile storage unit, comprises snapshot management means, inserted between the file system and the nonvolatile storage unit, for taking a snapshot which holds contents of the files at a predetermined timing in units of nonvolatile storage units and storing the snapshot in each nonvolatile storage unit, and snapshot control means for controlling read/write accesses with respect to the nonvolatile storage unit, and reference to the snapshot.
According to another aspect of the present invention, a computer system, which comprises recognition means for making the file system recognize a virtual disk drive storing files having contents held by the snapshot taken by the snapshot management means and stored on the nonvolatile storage unit, thereby allowing easy restart of a system using a given disk.
According to the present invention, a snapshot can be efficiently taken without modifying the existing operating system, file system, or application program, and the system can be easily restarted after a proper version of a disk image is selected.
The snapshot management means comprise snapshot taking timing means for controlling the taking timing of a snapshot, or snapshot taking instruction means for instructing taking of a snapshot.

5. [redacted]

[redacted]

The on-line storage devices on a computer are configured from one or more disks into logical units of storage space referred to herein as "containers." In prior systems, the containers are configured during the initial computer setup and can not be reconfigured during I/O processing without corrupting currently processing I/O requests. As storage needs on a computer system change, the system administrators may need to reconfigure containers to add disks to them or remove disks from them, partition disks drives to form new containers, and/or increase the size of existing containers. However, shutting down the system to reconfigure containers may be unacceptable for businesses.

One aspect of the system described herein is the routing of I/O requests in the I/O subsystem to a different container than previously pointed to by the operating system. Containers are created and maintained by a software entity called the "container manager." Each type of container on the system has an associated driver, which processes system requests on that type of container. After a complete backup operation, the backup program verifies the backed up files to make sure that the files on the secondary storage device (usually a tape) were correctly backed up. One problem with the backup process is that files may change during the backup operation.

To avoid backing up files modified during the backup process and to enable applications to access files during the backup operation, the container manager periodically (e.g. once a day) performs a procedure that takes a "snapshot" or copy of each read-write container whereby, the container manager creates a read-only container which looks like a copy of the data in the read-write container at a particular instant in time. Thereafter, the container manager performs a "copy-on-write" procedure where an unmodified copy of data in the read-write container is copied to a read-only backup container every time if there is a request to modify data in the read-write container. The container manager uses the copy-on-write method to maintain the snapshot and to enable backup processes to access and back up an unchanging, read-only copy of the on-line data at the instant the snapshot was created. This procedure is described in detail in related co-pending U.S. patent application Ser. No. 08/963,754, entitled "System and method for real-time data backup using snapshot copying with selective compaction of backup data".

During the backup procedure, the container manager creates a "snapshot" container, a "snapshotted" container and a "backing store" container. After the container manager takes the snapshot, the snapshotted container driver processes all input/output (I/O) requests, to store data in or retrieve data from a read-write container. The snapshotted container driver processes all I/O requests to retrieve data from the read-write container by forwarding them directly to the read-write container driver. However for all I/O requests to modify data in a read-write container, the container manager first determines whether the requested block of data has been modified since the time of the snapshot. If the block has not been modified, the container manager copies the data to the backing store container and then sets an associated bit-map flag in a modified-bit-map table. The modified-bit-map table contains a bit-map with each bit representing one block of data in the read-write container. After setting the modified-bit-map flag, the snapshotted container driver forwards the I/O storage request to the read-write container driver.

When the backup process begins execution, it invokes I/O retrieval requests from the snapshot container. In this example, a file system translates the file-oriented I/O request into a logical address and forwards the request to a snapshot container driver. The snapshot container driver checks the associated bit-map in the modified-bit-map table for the requested block of data. If the bit-map flag is set, the snapshot container driver forwards the request to the backing store container driver to retrieve the unmodified copy of that block from the backing store container. If the bit-map flag is not set, this means that the block has not been modified since the snapshot was created. The snapshot container driver forwards the request to the read-write container driver to retrieve a copy of that block of data from the read-write container. Upon retrieving the file from the backing store container or the read-write container, the backup process backs it up. After a complete backup operation, the container manager deletes the snapshotted container, the snapshot container, the backing store container, and the modified-bit-map table, and thereafter, forwards all I/O requests directly to the read-write container driver.

[redacted]

Ususally computer file systems have an archive bit to back up applications. If the archive bit is set up, the file will be backed up, otherwise not. The backup program clears the archive bit on each file after a complete backup of that file. If the file changes after the backup operation, the file system will set the value of the archive bit. Such approach allows backing up only of those file that were previously modifyed.

A problem with using only the archive bit is, if a file modified during the backup operation, the backup program clears the archive bit value incorrectly after a complete file backup, though the file was modified. If after this operation the file does not change, then its archive bit can not be set up and the file will no tbe backed up nex time.

To perform the backup operations a read-only snapshot container is created. The copy-on-write procedure is described in a co-pending U.S. patent application Ser. No. 08/963,754 entitled System and Method for Real-Time Data Backup Using Snapshot Copying with Selective Compaction of Backup Data. The snapshot container contains attributes of the file such as the value of the archive bit. The file contents in the snapshot container are preferably an identical copy of the file contents in the read-write on-line container at the instant the snapshot was taken. During the backup operation, the backup program backs up the file from the snapshot container and clears the archive bit. After a complete backup operation, the system deletes the snapshot container. However, the archive bit in the associated files in the read-write on-line container remains set and do not reflect the clear archive bit operation performed by the backup process. This may eventually lead to a situation where all files in the read-write on-line container have their archive bits set; thus the system performs a full system backup during every backup operation. The copy-on-write procedure resolves the productivity issue but it does not resolve the problem of an additional copy operation.

The invention comprises a method for enabling incremental backup operations by converting a backed up file ID to a read-write file ID and comparing an archive bit change number (ABCN) attribute in the read-write file ID with the ABCN attribute in the backed up file ID. The ABCN attribute is associated with each file and is manipulated by a file system of the computer to reflect a correct value of the archive bit when the file is modified during a current backup operation. Specifically, the novel ABCN attribute is incremented each time the file is modified to ensure that the file is accurately copied to secondary backup storage during a subsequent incremental backup operation. The method thus enables reliable on-line file modifications which, in turn, substantially increases the efficiency of the computer system.

According to the invention, the file system increments the ABCN on the read-write container when the file is modified. Instead of clearing the archive bit in a snapshot file after a backup operation, the file system converts a container ID in the snapshot file ID to a read-write container ID and locates the associated read-write file with the identical file ID as the snapshot file. Then it compares the ABCN in the read-write file with the ABCN in the snapshot file. If they match, the file system clears the archive bit in the read-write file. If they do not match, the archive bit is not cleared and the state of the file in the read-write container is correctl and the file will copyed during the next incremental backup operation.

[redacted]

A general purpose of the present invention is to provide for a new log device system architecture that maintains many of the advantages of conventional log structured file systems while providing a much wider range of support for different and concurrently executed application environments.

This is achieved through the use of a data storage subsystem that provides for the efficient storage and retrieval of data with respect to an operating system executing on a computer. The data storage subsystem includes a storage device providing for the storage of predetermined file and system data, as provided by the computer system, within a main filesystem layout established in the storage device. The data storage subsystem also includes a log device coupled in the logical data transfer path between storage device and the computer system. The log device provides for the storage of the predetermined file and system data within a log structured layout established in the log device. A control program, included as part of the data storage system, is executed in connection with the log device and provides system management over the log device to store the predetermined file and system data in one of a plurality of data segments, delimited by a first free data segment and an oldest filled data segment, and to selectively clear the segment in a given range and transfer the predetermined file and system data from the log device to the storage device. The control program utilizes location data provided in the predetermined file and system data to identify a destination storage location for the predetermined file and system data within the main file system. A log device file system is described in detail in related co-pending U.S. patent application Ser. No. 5,996,054, entitled Efficient virtualized mapping space for log device data storage system. Location of a log device in the architecture is described in U.S. patent application Ser. No. 5,832,515, entitled Log device layered transparently within a filesystem paradigm.

[redacted]

Computer systems often perform data backups on computer files to enable recovery of lost data. To maintain the integrity of the backed-up data, a backup process must accurately back up all files or back up all modified files after the most recent backup process. A backup program copies each file that is identified as a candidate for backup from an on-line storage device to a secondary storage device. On-line storage devices are configured from on one or more disks into logical units of storage space referred to herein as "containers". Containers are created and maintained by a software entity called the "container manager". Each type of container on the system has an associated driver which processes system requests on that type of container. After a complete backup operation, the backup program verifies the backed up files to make sure that the files on the secondary storage device(usually a tape) were correctly backed up. One problem with the backup process is that files may change during the backup operation.

To avoid backing up files modified during the backup process and to enable applications to access files during the backup operation, the container manager periodically (e.g. once a day) performs a procedure that takes a "snapshot" or copy of each read-write container whereby, the container manager creates a read-only container which looks like a copy of the data in the read-write container at a particular instant in time. Thereafter, the container manager performs a "copy-on-write " procedure where an unmodified copy of data in the read-write container is copied to a read-only backup container every time there is a request to modify data in the read-write container. The container manager uses the copy-on-write method to maintain the snapshot and to enable backup processes to access and back up an unchanging, read-only copy of the on-line data at the instant the snapshot was created.

During the backup procedure, the container manager creates a "snapshot" container, a "snapshotted" container and a "backing store" container. After the container manager takes the snapshot, the snapshotted container driver processes all input/output (I/O) requests, to store data in or retrieve data from a read-write container. The snapshotted container driver processes all I/O requests to retrieve data from the read-write container by forwarding them directly to the read-write container driver. However for all I/O requests to modify data in a read-write container, the container manager first determines whether the requested block of data has been modified since the time of the snapshot. If the block has not been modified, the container manager copies the data to the backing store container and then sets an associated bit map flag in a modified-bit-map table. The modified-bit-map table contains a bit map with each bit representing one block of data in the read-write container. After setting the modified-bit-map flag, the snapshotted container driver forwards the I/O storage request to the read-write container driver.

When the backup process begins execution, it invokes I/O retrieval requests from the snapshot container. A file system, which is a component of the operating system translates the file-oriented I/O request into a logical address and forwards the request to a snapshot container driver. The snapshot container driver checks the associated bit map in the modified-bit-map table for the requested block of data. If the bit map is set, the snapshot container driver forwards the request to the backing store container driver to retrieve the unmodified copy of that block from the backing store container. The backing store container driver then processes the backup process retrieval request. If the bit map is not set, this means that the block has not been modified since the snapshot was created. The snapshot container driver forwards the request to the read-write container driver to retrieve a copy of that block of data from the read-write container. Upon retrieving the file from the backing store container or the read-write container, the backup process backs it up. After a complete backup operation, the container manager deletes the snapshotted container, the snapshot container, the backing store container, and the modified-bit-map table and thereafter forwards all I/O requests directly to the read-write container driver.

The problem with the current copy-on-write process is that the read-write container and the backing store container must be the same size to maintain a fixed mapping between the read-write container blocks and the copied backing store container blocks. Usually, however, only a small amount of the on-line data is modified between backup operations, the present copy-on-write process therefore utilizes storage space inefficiently. Therefore, it is an object of the present invention to provide a system that allows copy-on-write procedures to be performed on a backing store container that is smaller than the read-write container while ensuring that the read-write container blocks are accurately mapped to the copied backing store container blocks.

In the backup system described herein, the container manager creates and maintains structures that map the unmodified copies of data in the backing store container to locations in the read-write container. The mapping structures which may be stored in memory and/or on disks, contain addresses of locations in the backing store container where collections of blocks of data are stored based on the original block address in the read-write container. In order to obtain the address of the location in the backing store container where a block of data is stored, the container manager converts the physical block number of the read-write block of data into a physical block address in the backing store container which actually contains the data. By using these mapping structures, the system provides an efficient manner of utilizing a smaller backing store container or less storage space, since the data modified during the snapshot backup process is usually substantially less than read-write on-line data.

Specifically, in the preferred embodiment of the invention, when the container manager creates a backing store container, it creates a level 1 table and stores the level 1 in memory and in the first space in the backing store container. When a block of data is copied from the read-write container to the backing store container, the container manager creates a level 2 table, a 2K longword space in the backing store container, and stores the block of data in the level 2 table. The container manager thereafter stores the beginning block number of the level 2 table in a level 1 table entry. The level 1 table is always resident in memory but, the level 2 table is cached in memory as blocks stored in that table is needed.

Before the container manager copies a block of data into the backing store container, it converts the physical block number in the read-write container into a virtual block number. The container manager uses bits zero to three of the virtual block number as the block offset, bits four to fifteen as the second index, and bits sixteen to thirty-one as the first index. The container manager utilizes the value of first index to access an entry in the level 1 table. It then checks to see if a level 2 table is available. If a level 2 table is not available the container manager creates a level 2 table; otherwise, it utilizes the available level 2 table. If the location of the block is outside of the existing level 2 table's access range, the container manager creates a new level 2 table. It thereafter stores the beginning block number of the level 2 table in the level 1 table entry. Then it uses the value of the virtual block number's second index to access an entry in a level 2 table and it adds the virtual block number's block offset to the level 2 table entry to determine the location where it will store the data on the backing store container.

During a backup operation, to retrieve data from the backing store container the container manager uses the first index in the virtual block number to access the level 1 table for the beginning block number of the level 2 table. It indexes the level 2 table with the second index in the virtual block number and adds the block offset in the virtual block number to the level 2 table entry. Then it reads the data from that location in the backing store container.

[redacted]

Summary of Invention

This invention provides a method of file system backup without suspending online application programs using file system snapshot. Such system significantly increases computer system availability and allows backing up without interrupting a service.

The computer file system is usually located on the block data storage and works with it at the level of blocks; that is, read-write is performed by the data areas having sizes divisible by the size of one block. The sequence of the blocks in the storage is ordered and each block has its own number. There may be several such storages and the file system may take only a part of one storage, the whole storage, or several such storages or their parts. On the disk such storages are usually located in so called partitions, taking the whole partition.

The given file systems can be usually subdivided into several categories – housekeeping data of the file system volume, file metadata, file data, and free space not occupied by the other data.

Servicing of of such a file system is provided by so called file system driver embedded into the operating system (Figure 1).

Usually, requests to read or write data to the data storage are initiated by OS user processes **100** or some OS thread processes **110** requesting some data exchange through the file system request.

This request usually comes to the file system driver **120**, which defines where in the data storage data blocks are located. Read and write of these data allows completing of the requested operation. Then the request comes to the OS cache **130** from where in case of absence of the requested data in cache or necessity to release space in cache using OS algorithms the above request is transmitted for execution to the driver OS storage driver **140** performing the operation on the data of the data storage **150**.

The data storage driver **140** usually works with the device in block mode; that is, it receives requests for data read and write using blocks of the fixed size. Each such block is supplied with the number according to which this operation is performed – this is a standard way of data addressing by such devices. Thus, the driver for the data write operation acquires a set of pairs (data block, number) for processing.

From the point of view of the file system data stored in the block data storage medium **200** all blocks can be subdivided into several classes (Figure 2). Depending on the file system type, they can store data specific for the volume, metadata of the file system **210**, file data **220**, or free space **230** not currently taken by other data. The most file systems consider that each data type takes its data block entirely and different data types cannot be combined in one block, though some file systems can combine these data under specific circomstances into one block (e.g., reiserfs or Microsoft Windows NTFS).

Thus, if you copy all data blocks that are not free (all besides entirely free blocks **230**) then you can get a so called file system snapshot to copy its state a a current moment of time.

Listing of file system blocks just taken is not a requirement and can be used to optimize the space taken by the backup procedure. In case it is impossible to get such information, the block fetching procedure may select absolutely all blocks participating in storing of any file system data, including free blocks.

Data backup operation is quite time consuming. Thus, to make backed up data correspond to any specific state at a given moment of time, it is required to provide that during backup data being copied do not change.

This tasl is not difficult if the data storage and a file system associated to it are not connected to any active computer or blocked from data modification; that is, there are no processes able to modify data (Figure 3). The data storage medium **300** must be re-written to the backup storage **330**. Copying of occupied areas **310** only without copying of free blocks **320** to increase performance and reduce space requirements is considered optimal. During backup of the file system data, subject to backup, are in the two different states: when they are already backed up **340** to the storage **360** and when they are not backed up yet, but only scheduled for backup 350. When backup is over, all data are located in the backup storage 370, and the file system and main data storage are ready to work.

If the file system is connected to an active computer and there are file system processes and user applications working with data during backup (so called on-line backup), then the task becomes more compicated (Figure 4). Such situation is typical for servers with high level of accessibility which cannot be stopped during backup.

Imaging that in the system a backup process of data of the block data storage **430** is launched so that it is performed within the off-line backup procedure. If a user process or file system process **400**, for example, disk cache, requested write to the data storage **410** to the storage device driver **420**, then it may happen that the required modification of data **470** and **480** should be done over data participating in backup **440**, and the modification process may request the data area **450** already copied to the backup storage **495** (request **470**) or data **460** which has not been copied yet (request **480**). Request **470** can be performed without without damaging backed up data, because backup is a one-pass process, which does not require return to the already processed areas. But request **480** can not be performed as the correctness of the backed up data can be affected and a modified block that does not belong to the given copy can penetrate into the the backup copy. This can make correct recovery of the file system state impossible, because data may refer to different points in time.

To solve this problem the current invention suggests using a temporary data storage container 490 intended for resolving of such situations and a specific procedure of data backing up into a backup storage.

The given procedure operates at the level of underlying file system blocks and can be used as a data storage, working with a file system designed using a block principle. The only requirement to the system is that there must be a procedure to define by the number of a block that it belongs to data and metadata of the file system. For internal purposes, the backup procedure is to efficiently define which blocks are subject to copy and whether any given block was copied or not.

The intermediate data storage container is a temporary buffer with the block design of the data storage and can be located both on the external regarding the backed up data storage space **490** and on a dedicated part of the storage, which can reperesent a separate partition of storage **240** or reserved for this purpose file within storage **250**.

The on-line backup procedure begins with informing the driver of the operating system, servicing the data storage, that the data are in the backup state. Optionally, a procedure of the operating system cache reset can be called to write so-called "dirty" pages into the data storage before informing the driver about the backup start, which increases actuality of the data stored in the snapshot. Then the list of data storage blocks is compiled. This list contains file system data of different types that should be backed up. Then the backup procedure begins; it takes blocks from the list and copies them into the backup storage. On completion of each block backup it is marked as backed up. During the backup process blocks must remain invariable. When the last block is backed up or the backup procedure is canceled which can be induced by the appearance of fatal errors or by the user's decision or by the processes of the operating system, the OS driver servicing the data storage is informed about the procedure completion and continues functioning in the customary mode.

The OS driver, servicing the data storage, must provide an opportunity to communicate with the backup procedure. Since the backup procedure started this driver is to provide it with the data block numbers requested for write into the data storage by the opearting system or a user process.

The backup procedure depending on the state of its internal data is responsible whether each requested block was copied to the backup storage or not. If the block was not copied, then the OS driver suspends the block write and waits when it is released.

The requested block can be released by continuing the backup procedure; that is, when its turn comes. This means that the request processing time can be very long and usually such mode is unacceptable for online systems.

The other variant of the block release is associated with the usage of a specific container used as an intermediate data storage (Figure 5). When the data storage driver gets a request to write a block into the area already copied by the backup procedure **510**, the required write is performed without peculiarities and limitations. If the incoming request requests the write to an area not yet backed up **500**, then the write process is suspended, the current state of the given data area is copied to the intermediate data storage container **520**, and only when the copy procedure is over the write procedure **500** is allowed. Thus, the content of the data block at the moment when the backup procedure started is stored in container **530**, from where it will be copied by the backup procedure when it is required **540**. Thus, delays when writing to the main storage are reduced to minimum and the programs running on the computers conncted to the data storage can continue working practically without pauses.

Data from the intermediate storage container can be re-written to the backup storage when the write procedure of data from the main storage was over (Figure 6). In this situation despite the backup process from the main storage is completed and requests for write to any regions of the data storage 600 are performed by the driver immediately, there remains a need to write data temporarily stored in the data container 610 to the backup storage 620. Thus, an additional write and usual computer activity may happen in the concurrent mode depending on the backup data storage.

If during the write process to the temporary container it gets overfilled (Figure 7), then the processes of data write to the main storage to the unsaved area 700 should be stopped, and the temporary data in container 720 should be re-written to the backup container 730 to free the space for further records. Though even if in the given moment there are some requests for write to the main data storage 710 over the data already written, their execution should not be stopped.

This invention differs from its analogues in the following way:

From file system Episode and its methods of snapshot creation the given invention differs in that it works at the level of blocks of the disk data storage and not at the level of inodes or files; besides that it is based on a specific container as a means of intermediate data storage until it is stored in the backup storage.

From file system WAFL and its methods of snapshot creation the given invention differs in that it works at the level of blocks of the disk data storage and not at the level of inodes or files.

[redacted] given invention [redacted] does not use file search paths and the created snapshot is a set of blocks and not files.

[redacted] the given invention [redacted] works at the level of blocks of the disk data storage and not at the level of inodes or files; and the created snapshot is a set of blocks and not files.

[redacted]f [redacted] the given invention [redacted] works at the level of blocks of the disk data storage and not at the level of inodes or files; it does not use tables of pointers to virtual and logical addresses but consequently copies blocks filled with file data to backup data storage and, also, it has a specific procedure to fetch blocks participating in the snapshot and management at the level of the driver of the data storage medium.

[redacted] the given invention [redacted] does not specifically store the data allowing to identify a virtual disk and does not undertake any specific steps to mount the file system directly from tha snapshot and, also, it has a specific procedure to fetch blocks participating in the snapshot and management at the level of the driver of the data storage medium.

[redacted] the given invention [redacted] uses only one container and only for intermediate temporary storing of data requested for modification by the OS processes at the moment of the backup procedure; the given invention uses a specific procedure to fetch blocks participating in the snapshot creation which can avoid copying the blocks not occupied by the file system and also readdressing of all read-write calls to the snapshot container [redacted] is not performed; and in the given patent requests for data exchange coming from the the data storage driver are always performed on the data stored in the data storage. [redacted]

[redacted] the given invention [redacted] works at the level of blocks of the disk data storage and not at the level of files and archive bit used as an indication that archiving is necessary.

[redacted], the given invention [redacted] is not oriented to support a single journaling file system and has specific means to support online data backup.

[redacted] the given invention [redacted] works at the level of blocks of the disk data storage and not at the level of files and archive bit used as an indication that archiving is necessary.

Figures

Figure 1. User and OS kernel thread processes write data into the data storage using the OS file system driver that can use the OS cache to write data.

Figure 2. Block data container, temporarily storing some data intended for backup, can be located on an external medium, on a continuous partition of the data storage or inside of the file system file.

Figure 3. Process of the consecutive write of the file system data during a usual (not online) data backup procedure.

Figure 4. Process of the consecutive write of the file system data during the online data backup procedure when a request for write over the data already or not yet copied can emerge within the system.

Figure 5. Write procedure to the area of data already copied can be performed directly; before the write to the area of data not yet copied, old data from that area are preliminary copied to the block data container and only then the write to the area not yet copied is allowed.

Figure 6. Data from the block data container are copied to the backup storage at any convinient moment.

Figure 7. In case the block data container is overfilled, the data write process to it is blocked and its data are written to the backup storage.

User and OS kernel thread processes write data into the data storage using the OS file system driver that can use the OS cache to write data.

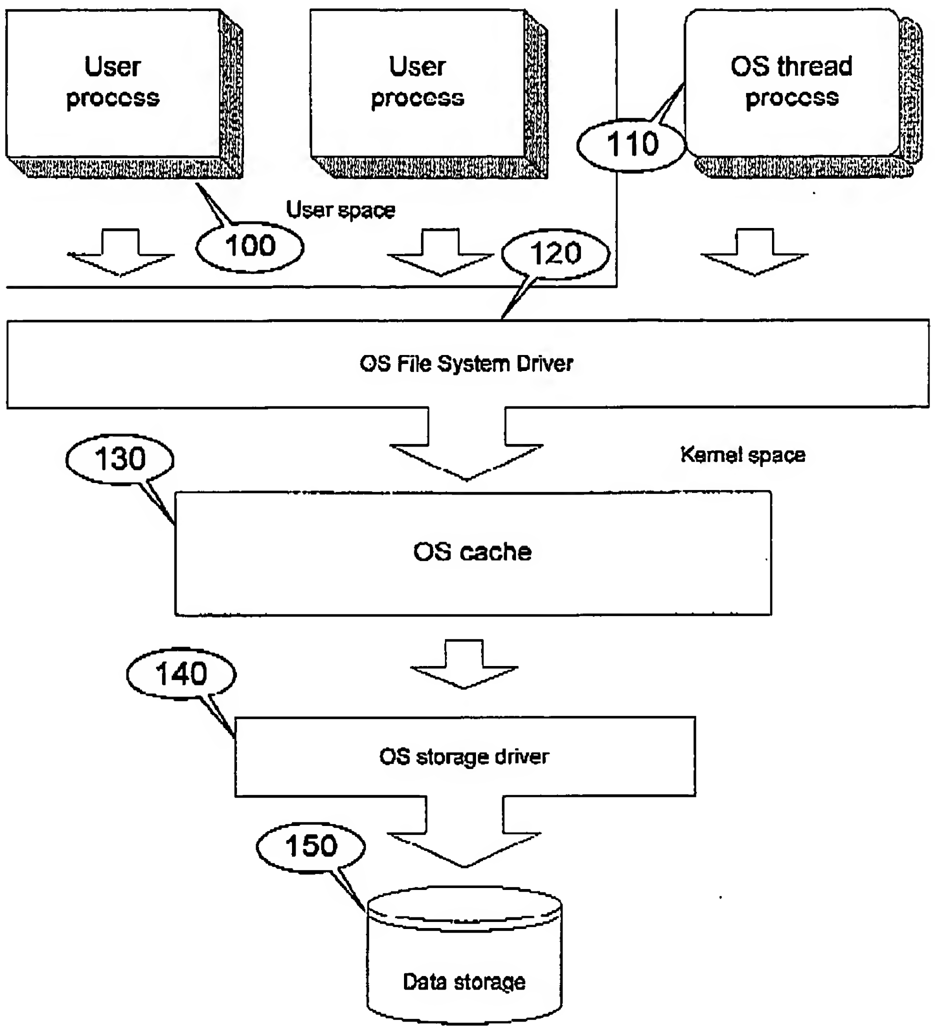


Figure 1.

Block data container, temporarily storing some data intended for backup, can be located on an external medium, on a continuous partition of the data storage or inside of the file system file.

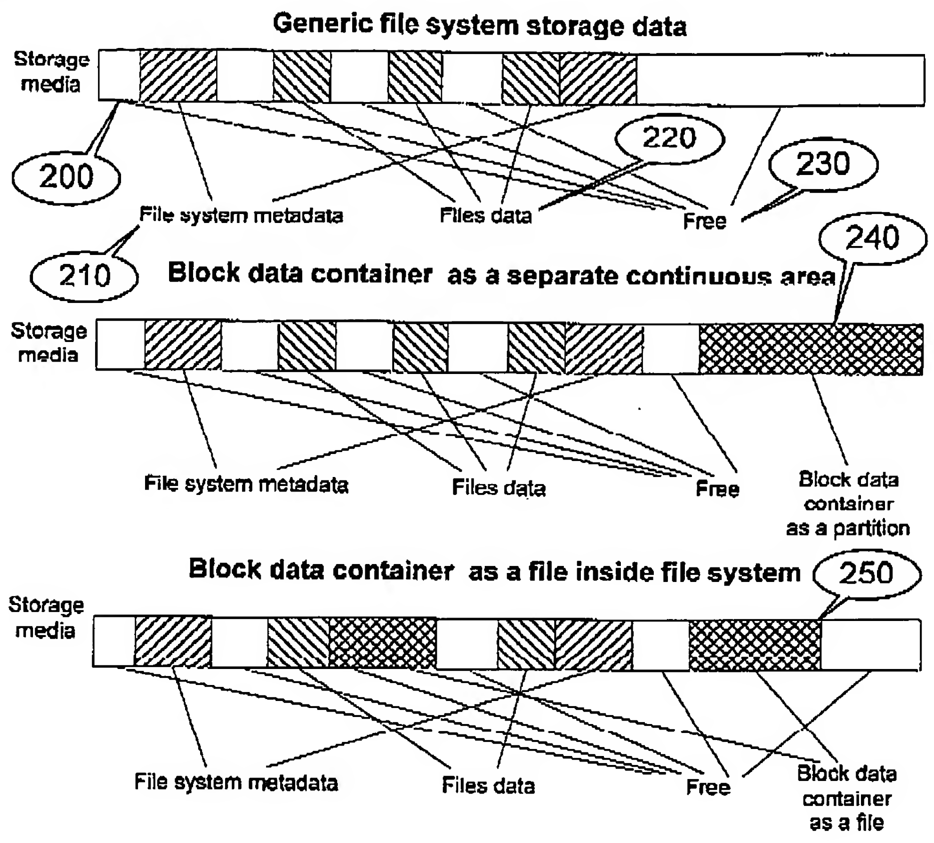
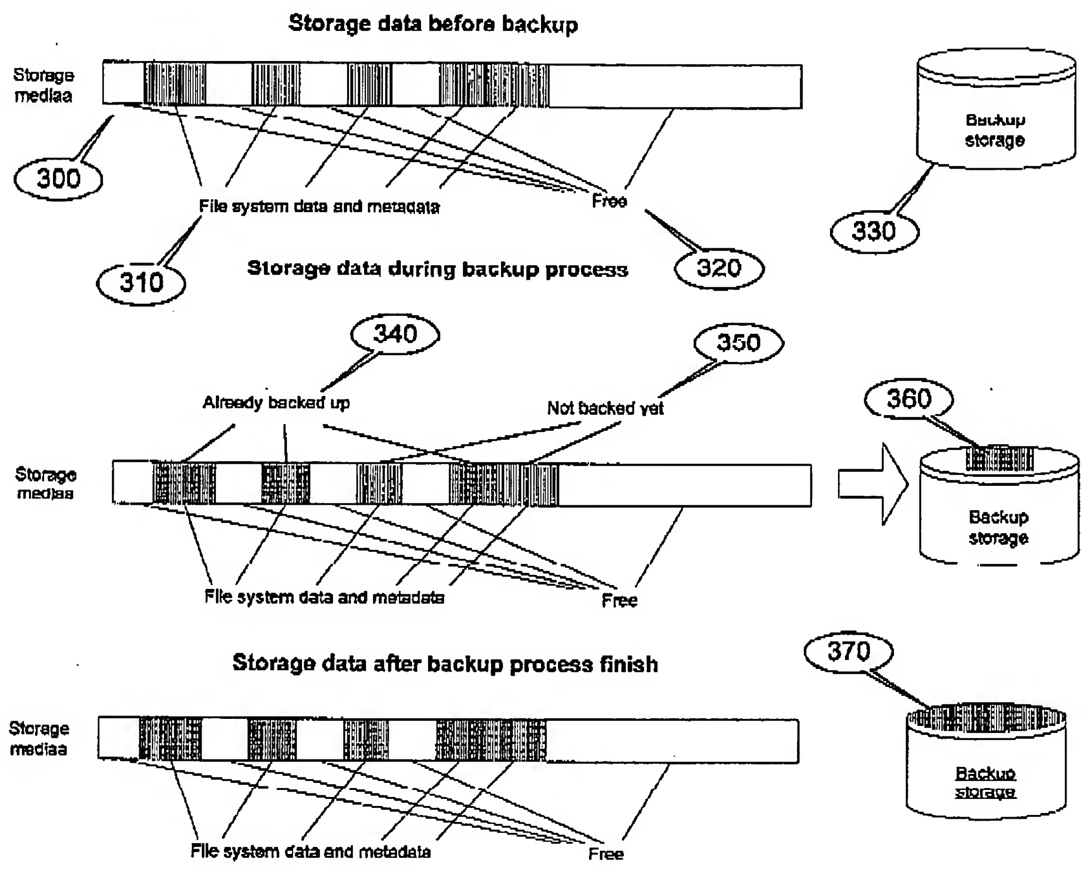

Generic file system storage data
Storage media
200
220
230
File system metadata
Files data
Free
210
240
Block data container as a separate continuous area
Storage media
File system metadata
Files data
Free
Block data container as a partition
Block data container as a file inside file system
250
Storage media
File system metadata
Files data
Free
Block data container as a file

Figure 2.

Process of the consecutive write of the file system data during a usual (not online) data backup procedure.



Storage data before backup
Storage mediaa
Backup storage
300
File system data and metadata
Free
330
310
Storage data during backup process
320
340
350
Already backed up
Not backed yet
360
Storage mediaa
Backup storage
File system data and metadata
Free
Storage data after backup process finish
370
Storage mediaa
Backup storage
File system data and metadata
Free

Figure 3.

Process of the consecutive write of the file system data during the online data backup procedure when a request for write over the data already or not yet copied can emerge within the system.

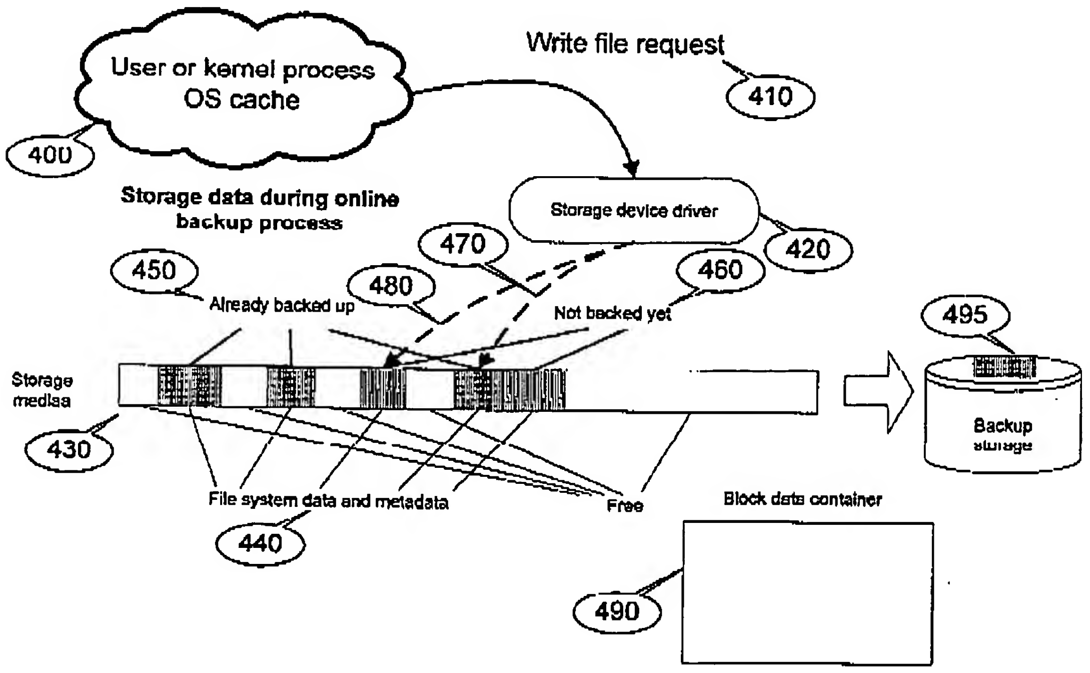


Figure 4.

Write procedure to the area of data already copied can be performed directly; before the write to the area of data not yet copied, old data from that area are preliminary copied to the block data container and only then the write to the area not yet copied is allowed.

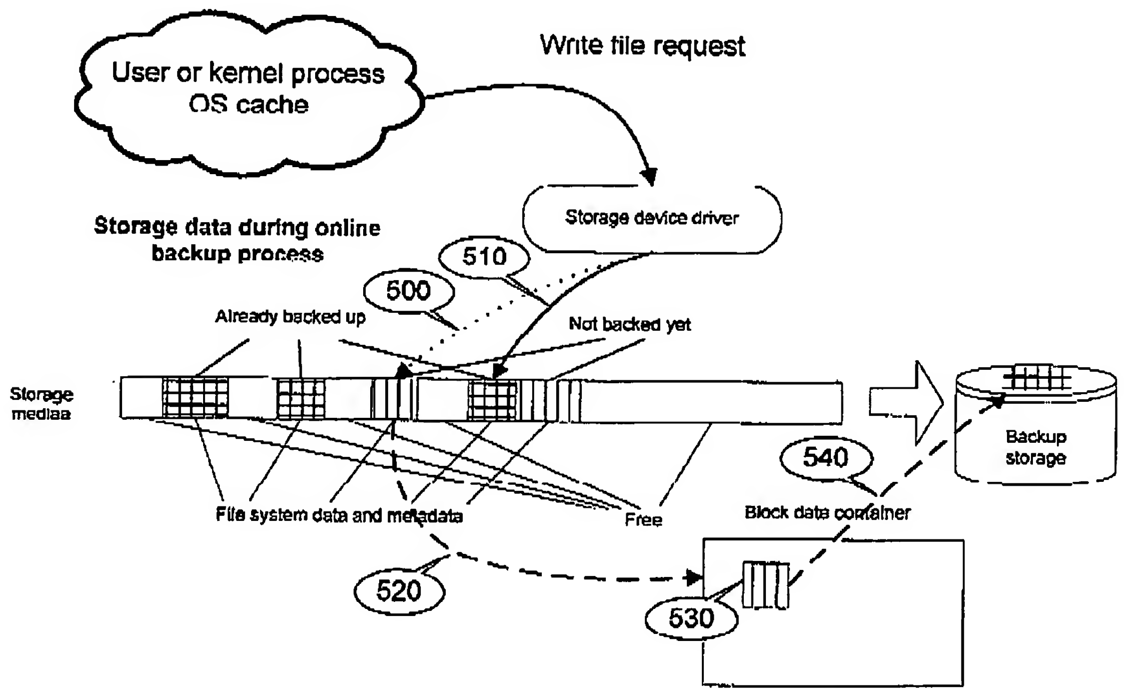


Figure 5.

Data from the block data container are copied to the backup storage at any convinient moment.

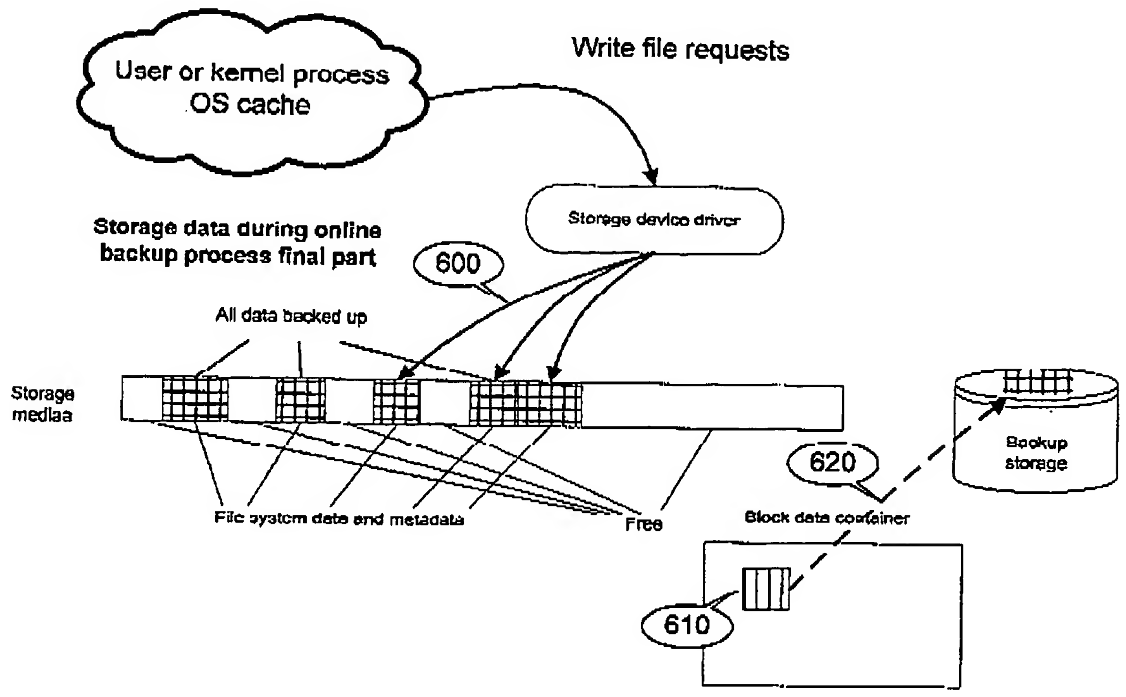


Figure 6.

In case the block data container is overfilled, the data write process to it is blocked and its data are written to the backup storage.

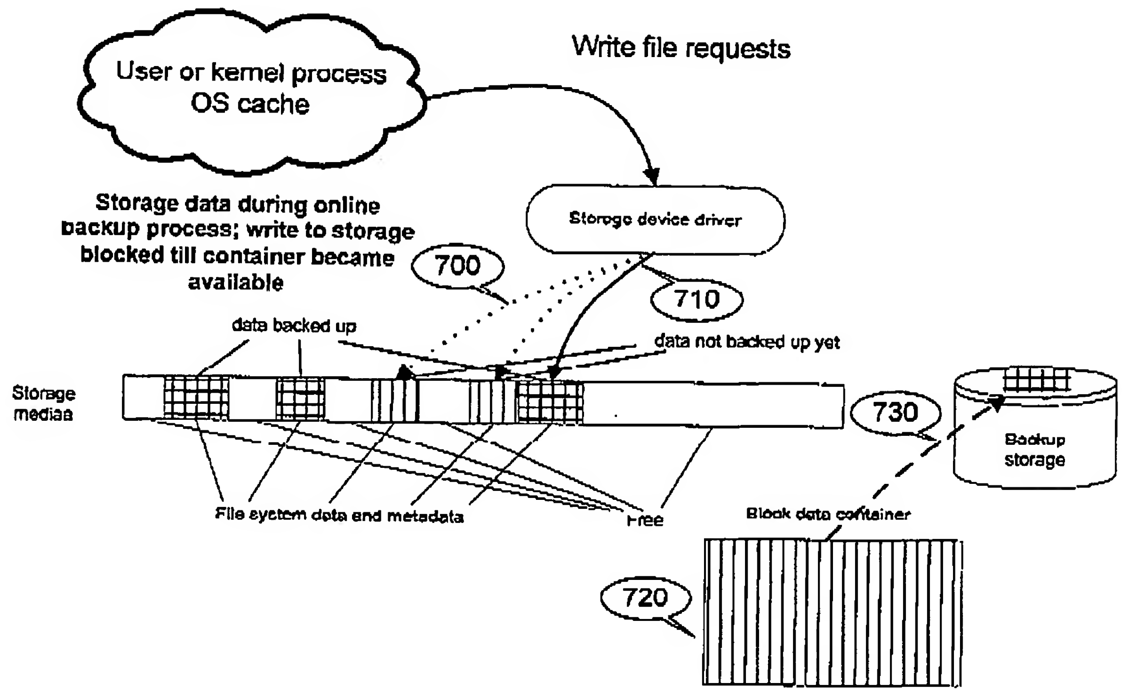


Figure 7.

RECEIVED
CENTRAL FAX CENTER
DEC 2 2 2005

EXHIBIT B

O'MELVENY & MYERS LLP
400 SOUTH HOPE STREET
15TH FLOOR
LOS ANGELES, CA 90071

August 12, 2003

SERGUEI BELOUSSOV
SWSOFT
395 OYSTER POINT BLVD., SUITE 213
SOUTH SAN FRANCISCO, CA 94080

INVOICE NUMBER: 531960
MATTER NUMBER : 0848265-00002

Requesting Attorney: BRIAN BERLINER

Tax Identification No: 95-1066597

FOR PROFESSIONAL SERVICES RENDERED THROUGH JULY 31, 2003

USA/METHODS OF USING FILE SYSTEM SNAPSHOTS FOR +

DATE	NAME	DESCRIPTION	HOURS
Group: Attorney			
07/21/03	B BERLINER	REVISE/EDIT DRAFT PATENT APPLICATION	.40
07/22/03	B BERLINER	REVIEW PATENT APPLICATION PRIOR TO FILING WITH U.S. PATENT AND TRADEMARK OFFICE	.30
* * Subtotal:		ATTORNEY HOURS	.70
Group: Document Production			
07/22/03	A SUTANANDI	FILING OF PATENT APPLICATION WITH THE U.S. PATENT AND TRADEMARK OFFICE	1.00
* * Subtotal:		DOCUMENT PRODUCTION HOURS	1.00

TOTAL CHARGEABLE HOURS -- 1.70

FEES -- [redacted]

O'MELVENY & MYERS LLP
400 SOUTH HOPE STREET
15TH FLOOR
LOS ANGELES, CA 90071

July 10, 2003

SERGUEI BELOUSSOV
SWSOFT
395 OYSTER POINT BLVD., SUITE 213
SOUTH SAN FRANCISCO, CA 94080

INVOICE NUMBER: 527241
MATTER NUMBER : 0848265-00001

Requesting Attorney: BRIAN BERLINER

Tax Identification No: 95 1066597

FOR PROFESSIONAL SERVICES RENDERED THROUGH JUNE 30, 2003

USA/SYSTEM AND METHOD FOR PROVIDING FILE-SHARING

DATE	NAME	DESCRIPTION	HOURS
Group: Document Production			
06/04/03	MZ CHUA	PREPARE DOCUMENTS & SENDING SAME TO CLIENT NECESSARY TO RESPOND TO NOTICE TO FILE MISSING PARTS	1.00
* * Subtotal:		DOCUMENT PRODUCTION HOURS	1.00

TOTAL CHARGEABLE HOURS -- 1.00

FEES -- [redacted]

SUPPORT SERVICES AND CHARGES

COPYING .90

TOTAL SUPPORT SERVICES AND CHARGES ---------------------------- 0.90

TOTAL FOR: 0848265-00001 -- [redacted]

George Bardmesser

From:	Berliner, Brian [BBerliner@OMM.com]
Sent:	Wednesday, June 25, 2003 12:44 PM
To:	'Alexander Tormasov'
Subject:	RE: File System Snapshots for Data Backup (8482654-2)
Attachments:	SWsoft power of attorney doc

Attached is a sample power of attorney form. Note that the form must be modified to insert the filing date, serial number and title for each application. Also, there must be correspondence between the company name issuing the power of attorney and the assignment documentation for the patent application. If there is no assignment documentation, then the company lacks authority to issue the power of attorney.

-----Original Message-----
From: Alexander Tormasov [mailto:tor@sw-soft.com]
Sent: Monday, June 23, 2003 5:50 AM
To: Berliner, Brian
Subject: Re: File System Snapshots for Data Backup (8482654-2)

ok. can you send me this sample?
should it be independently filled for every patent?

----- Original Message -----
From: "Berliner, Brian" <BBerliner@OMM.com>
To: "'Alexander Tormasov'" <tor@sw-soft.com>
Sent: Friday, June 20, 2003 10:02 PM
Subject: Re: File System Snapshots for Data Backup (8482654-2)

> Thanks. The preparation and filing of power of attorney documents is
simple
> and inexpensive (probably less than $50 per document). We can send you a
> sample in electronic form.
>
> -----Original Message-----
> From: Alexander Tormasov
> Sent: Fri Jun 20 09:42:17 2003
> To: Berliner, Brian
> Subject: Re: File System Snapshots for Data Backup (8482654-2)
>
> I (and other inventors) sign this and it is ready to be send to you.
> I have another thing to do - about taking power of attorney for some of
our
> patents.
> before this I need to evaluate approximate price for this operation, and
> related costs in the future
> (for correspondence with USPTO).
> Also, I need a sample document to sign to send to USPTO to give OOM such a
> responsibility for each patent
> (it should be about 10 of them).
>
>
> ----- Original Message -----
> From: "Berliner, Brian" <BBerliner@OMM.com>

> To: <tor@sw-soft.com>
> Sent: Wednesday, June 11, 2003 9:36 PM
> Subject: FW: File System Snapshots for Data Backup (8482654-2)
>
>
>>
>> The declaration document is attached. bmb
>>
>> -----Original Message-----
>> From: Alexander Tormasov [mailto:tor@sw-soft.com]
>> Sent: Wednesday, June 11, 2003 10:03 AM
>> To: Berliner, Brian
>> Subject: Re: File System Snapshots for Data Backup (8482654-2)
>>
>>
>> please, send me a .doc file.
>> I am (and other inventors) are in Russia in this moment;
>> I will print it, sign and send you via FedEx.
>>
>> ----- Original Message -----
>> From: "Berliner, Brian" <BBerliner@OMM.com>
>> To: "'Alexander Tormasov'" <tor@sw-soft.com>
>> Sent: Wednesday, June 11, 2003 8:36 PM
>> Subject: RE: File System Snapshots for Data Backup (8482654-2)
>>
>>
>>> It was sent by overnight mail, not email.
>>>
>>> -----Original Message-----
>>> From: Alexander Tormasov [mailto:tor@sw-soft.com]
>>> Sent: Wednesday, June 11, 2003 9:35 AM
>>> To: Berliner, Brian
>>> Subject: Re: File System Snapshots for Data Backup (8482654-2)
>>>
>>>
>>> seems that I did not receive it...
>>> I check all emails May 29 - no attachements found...
>>> Could you resend declaration text again?
>>>
>>> ----- Original Message -----
>>> From: "Berliner, Brian" <BBerliner@OMM.com>
>>> To: "'Alexander Tormasov'" <tor@sw-soft.com>
>>> Sent: Wednesday, June 11, 2003 7:21 PM
>>> Subject: RE: File System Snapshots for Data Backup (8482654-2)
>>>
>>>
>>>> Tor:
>>>>
>>>> We sent the declaration to you, Serguei Beloussov and Stanslav
> Protassov
>>> on
>>>> May 29, 2003, and have not received the signed paper back yet. We
> have
>>> been
>>>> waiting for this paper before filing the patent application. Let me
>> know
>>>> the status.
>>>>
>>>> Thanks,
>>>> bmb
>>>>

REMAINING PAGES REDACTED

George Bardmesser

From: Alexander Tormasov [tor@sw-soft.com]
Sent: Wednesday, June 11, 2003 1:03 PM
To: Berliner, Brian
Subject: Re: File System Snapshots for Data Backup (8482654-2)

please, send me a .doc file.
I am (and other inventors) are in Russia in this moment;
I will print it, sign and send you via FedEx.

----- Original Message -----
From: "Berliner, Brian" <BBerliner@OMM.com>
To: "'Alexander Tormasov'" <tor@sw-soft.com>
Sent: Wednesday, June 11, 2003 8:36 PM
Subject: RE: File System Snapshots for Data Backup (8482654-2)

> It was sent by overnight mail, not email.
>
> -----Original Message-----
> From: Alexander Tormasov [mailto:tor@sw-soft.com]
> Sent: Wednesday, June 11, 2003 9:35 AM
> To: Berliner, Brian
> Subject: Re: File System Snapshots for Data Backup (8482654-2)
>
>
> seems that I did not receive it...
> I check all emails May 29 - no attachements found...
> Could you resend declaration text again?
>
> ----- Original Message -----
> From: "Berliner, Brian" <BBerliner@OMM.com>
> To: "'Alexander Tormasov'" <tor@sw-soft.com>
> Sent: Wednesday, June 11, 2003 7:21 PM
> Subject: RE: File System Snapshots for Data Backup (8482654-2)
>
>
> > Tor:
> >
> > We sent the declaration to you, Serguei Beloussov and Stanslav Protassov
> on
> > May 29, 2003, and have not received the signed paper back yet. We have
> been
> > waiting for this paper before filing the patent application. Let me know
> > the status.
> >
> > Thanks,
> > bmb
> >
> [redacted]
> >
> >

REMAINING PAGES REDACTED